MISSION CRITICAL

# Ontology for the Digital Enterprise

**From legacy complexity to an agile business-driven digital platform**

## 1. Digital challenge, software challenge

Performance and quality of hardware (servers, storage, networks, clouds, tablets, smart-phones) are continuously improving while prices are decreasing.

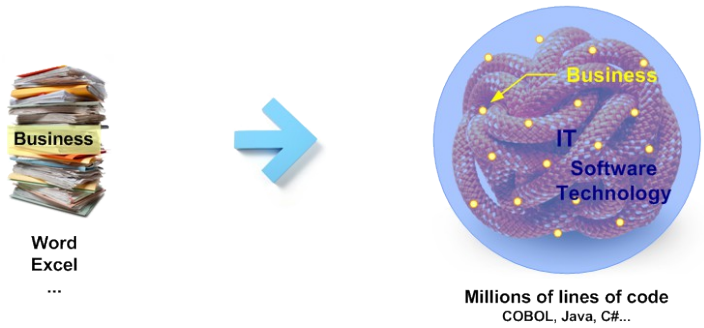
It's a complete different story when you look at **mission-critical enterprise software**. Developed over decades on mainframes, in COBOL or PL/1, its maintenance consumes up to 85% of IT budgets. Made worse by the current economy forcing businesses to seriously cut costs.

Innovating in these conditions is very challenging, more so as Business puts pressure on IT on multiple fronts: agility, single view of customers, cross-selling, intelligent use of social networking data, integrating customers and partners into the digital enterprise, with a quality on par with the best consumer web applications), etc.

**Software** is now a major **challenge**: TCO (Total Cost of Ownership) of mission-critical applications[1] must be reduced dramatically, while flexibility and quality[2] must be greatly improved.

### 1.1 Origin of this challenge

With the current approach (being waterfall or agile), developers **intertwine**, in the same program, the **business knowledge** and the **implementation details**: business specific knowledge is drowned in millions of technical lines of code:




#### First Challenge: the "What"

In his famous book *The Mythical Man-Month*[3], Fred Brooks wrote: "*The hardest single part of building a software system is deciding precisely what to build*".

Additionally, business people find it difficult to describe their requirements with the precision and completeness required by developers. Moreover, when they innovate, they are in a creative process, discovering the problem while the software is being built. As a consequence, they make many change requests, putting pressure on the developers.

[1] As opposed to ERP and productivity software (Office...)
[2] As an example, see http://www.cio.com/article/564613/5_Reasons_Businesses_Still_Hate_Enterprise_Software
[3] http://en.wikipedia.org/wiki/The_Mythical_Man-Month

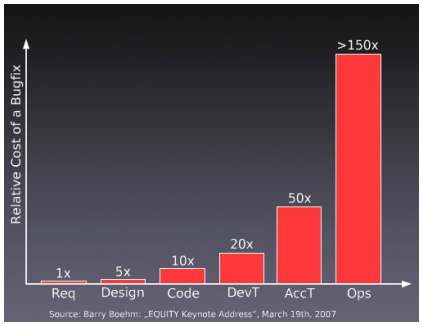
### Second Challenge: the "How"

Software development is complex, error prone and expensive: investment and maintenance costs are high, deadlines are difficult to keep, quality is challenging and it can be very hard to adjust the application software as business changes. The CHAOS study from the Standish Group, and many other articles, report on these issues:




### Third Challenge: the "What" and the "How" intertwined

When the "What" and the "How" are intertwined (developers express the business knowledge with a programming language like COBOL, PL/1, Java, C#, Python), the *essential* complexity of the **problem** and the *accidental* complexity of the technical **solution** are combined: the developer has to cope with a **complexity to the square**. As the problem remains insufficiently defined, business changes impact many lines of code, with unpredictable side effects requiring a lot of fixing. This explains why an error identified during acceptance testing is **50 times** costlier to fix than the same error identified during the specification phase. And more changes increase the **technical debt**, the famous "spaghetti" effect.




## 1.2 How to deal with these challenges?

First, the **specification** (the definition of the problem) must be separated from the **implementation** (its IT solution) in order to:

- deal **independently** with the problem complexity and the technical complexity, avoiding the multiplying effect (complexity to the square);
- use **different languages**: one well fitted to expressing the business knowledge and another adapted to computer programming (such as Java or C#).

The business description language must be **declarative**, i.e. only express the "what", and not the "how" (what classical imperative languages are good at).

The **specification** has to be more than just documentation: expressing the essence of the business problem, it **must drive the application**. The program and the specification are then always in sync.

Finally, as for any modern industrial processes, automation is required to reduce costs and delays, and deliver consistent quality and flexibility.

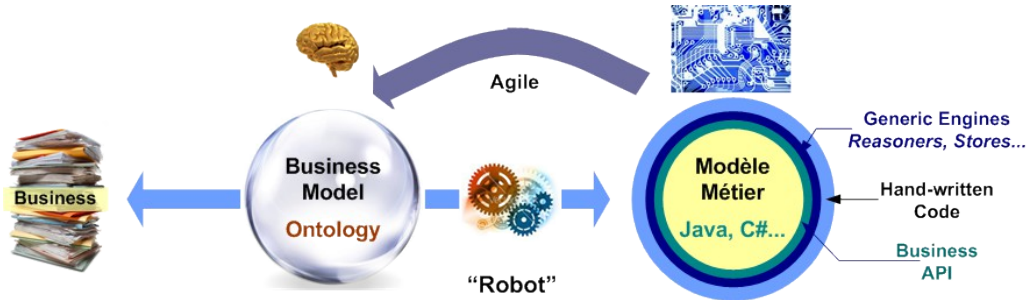
## 1.3 MCit ontology aproach

MCit has the technology which addresses all these challenges. This new way of developing software is based on very innovative concepts which are at the heart of the **W3C** (*World Wide Web Consortium*) *Semantic Web*[4].

It is structured around the following principles:

[4] http://www.w3.org/standards/semanticweb/

- Use of science – theories, models, software mathematics (formal logic) – to move from the current craft (building bridges like the Romans did) to a scientific approach ("computeing" the software like we engineer a bridge before constructing it).
- Model the business problem by creating an ontology[5]: an accurate, concise and executable representation of the **whole "what"** (concepts, rules, processes and data describing all the business logic), **completely separated from the technical "how"**.
- Possibility to **test**, **explain**, **verify** and **validate** the model before writing a single line of code (because specifications are executable).
- Aggressively reduce development costs thanks to **"robots"** that automatically generate a programmatic version of the ontology in the target programming language (the development process is industrialized).
- Developers use the business model through a **business API**[6]: they develop what is not in the model (and what should not be in the model, to prevent polluting the "what" by the "how"). For example the UI, DB, SOA[7], middleware[8], and integration are not part of the model. IT are able choose the **architecture**, **technology**, **frameworks** and **tools** used.
- Developers have access to a **generic software execution platform** containing all the components (independent from the application logic) required by any application. This platform – like a DVD player "playing" the ontology – significantly reduces development costs and increases the reliability and performance of the application by reusing optimized generic components (reasoners, stores, process engine, etc.). Thanks to automatic parallelization, it is ready for future dense multicore processors.
- It is easy to **change the application**: the ontology is changed to the new requirements and the application is **regenerated**. Quality results from the logic continuum, from the ontology to the software driven by this ontology.
- The application is developed in an **agile** way, starting from incomplete specifications. With each iteration, the ontology is enriched, validated by business experts and the application is regenerated. Going **from an idea to** a running **software** is very **fast**.
- As the ontology is independent from the programming language, it is possible to reuse the same ontology with other implementation and execution technologies.




## 1.4 ODASE

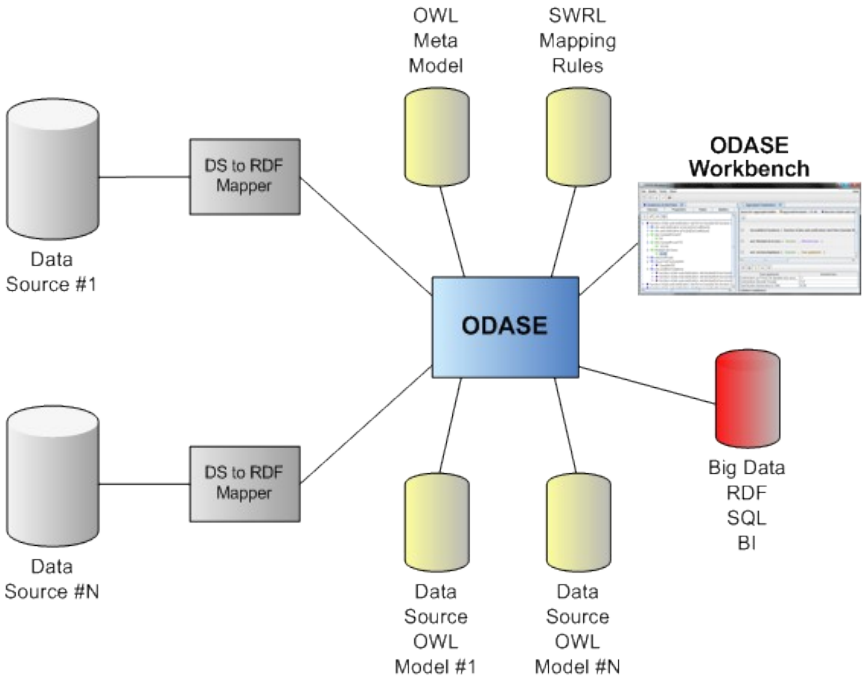
**ODASE**™ (*Ontology-Driven Architecture and Service Engineering*), is the name given by MCit to its platform and tools that leverage ontologies to rapidly build flexible, high-quality and cost-effective applications.

ODASE ontologies use the standards defined by the W3C Semantic Web[9] a long term viability guarantee.

[5] http://en.wikipedia.org/wiki/Ontology_(information_science)
[6] Application Programming Interface
[7] User Interface, DataBase, Service oriented Architecture
[8] http://en.wikipedia.org/wiki/Middleware_(distributed_applications)
[9] http://www.w3.org/standards/semanticweb/

### 1.5 ODASE and Big Data

*Semantic Web Technologies* are central to ODASE. They bring the required intelligence to exploit a web of data (such as *Open Data*). They also make it much easier to have an integrated view on data traditionally isolated in silos (e.g. Single View of Customer), for transactional, event-driven systems or *Business Intelligence* :




## 2. MCit Products and Services

- ODASE, platform and tools
- ODASE support and maintenance
- Ontology development and support services
- ODASE development and support services
- Studies : modernization, business architecture, pilot project, semantic BigData, etc.

## 3. Some frequently asked questions

Q – Is the construction of an ontology difficult for a business analyst?

No. Enterprise developers, traditionally with a COBOL background, are good business analysts capable of creating an ontology. Our experience shows that their move from COBOL to an ontology development paradigm is much easier than a transition from COBOL to Java which, being purely technical, does not add any business benefit.

Q – Is the complexity of the development process simply shifted from implementation to modelling?

No. Modelling delivers a very concise, precise and formal ontology which can be tested, explained and verified. Fixing an error in the ontology requires understanding of only a small set of knowledge elements (concepts, properties, axioms, rules), expressed using familiar business terms which are combined using clear logical relationships.

Q – What amount of work is spent on modelling vs implementation?

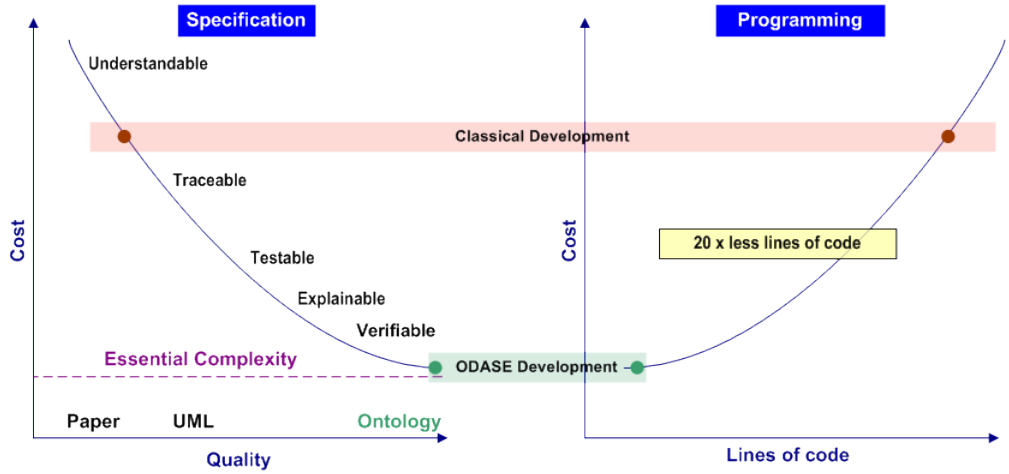
Around 50% of the total effort is devoted to the business side of the development: the agile construction and validation of the ontology. The remaining 50% is spent on the implementation. Compared to a classical development, the number of hand-written lines of code is reduced at least by a factor 20, with a dramatic impact on cost, quality, and time.

Q – Is it possible to use that approach in a step-by-step manner (not a Big Bang)?

Yes, and it should be done in that way. *"A complex system that works is invariably found to have evolved from a simple system that worked. The inverse proposition also appears to be true: A complex system designed from scratch never works and cannot be made to work. You have to start over, beginning with a working simple system."* (John Gall, *Systemantics*). The ontology can be extended and refined iteratively.

Q – Why is the cost reduction so dramatic?

This results from the combination of three points: (1) a complete specification of the business problem, tested and verified, (2) 0 defect automatic code generation and (3) much less hand-written code:




# 4. Conclusion

The clean separation between the business definition (ontology) and the implementation (programs) , combined with automatic code generation and programs driven by the ontology, delivers major **benefits**:

- Much lower cost, for development and maintenance
- Fast transition from ideas (in the ontology) to an effective application
- Business model tested, explained and verified before the first line of code is written (reducing risk)
- Business changes can be taken into account rapidly without destabilizing the existing code
- Better balance of work bewteen Business and IT
- Single point of business definition shared by Business and IT
- The business definition is always up-to-date as it drives the application
- Explicit business knowledge, not lost in technical code
- Low complexity : expressed with only a few hundred concepts, rules and processes
- IT can rely on the precise and clear semantics of the ontology
- Much less hand-written code, at least 20 times less
- Changes in the ontology are immediatley reflected in the code
- Business knowledge is formalized independently of a programming language
- This knowledge can be reused with new technologies : it is a reusable asset, never lost
- Approach based on open standards (W3C), using web technologies
- Semantic *Big Data* delivers knowledge
- Usable on-site and in the *cloud*

For additional information, contact info@missioncriticalit.com